Jogos isométricos em dispositivos móveis

Fernando Antonio Mota Trinta[1]
Oscar Gomes de Miranda[1]
Geber Lisboa Ramalho [1]

[1] Centro de Informática – Universidade Federal de Pernambuco
{famt, ogm, glr}@cin.ufpe.br

RESUMO

Este trabalho apresenta em linhas gerais os procedimentos realizados para extensão do wGEM - *Wireless Game Engine for Mobile Devices*, um motor para criação de jogos em dispositivos celulares, visando o suporte de jogos isométricos. Neste artigo é primeiramente apresentado o wGEM, focando no contexto de seu surgimento, objetivos e tecnologias utilizadas. Em seguida, é introduzido o conceito de jogos isométricos, ressaltando as dificuldades de implementação dos mesmos. Por fim, são apresentados os pontos mais relevantes do projeto de extensão do wGEM de modo a suportar jogos isométricos, onde, por fim é apresentado como validação do trabalho, o jogo "*The Treasure Hunt*", que utiliza as funcionalidades implementadas no projeto.

Palavras Chave: *Jogos Isométricos, Java 2 MicroEdition, Engenharia de Software*

---

1. Introdução

O sucesso de jogos eletrônicos em computadores pessoais e vídeo games faz com que atualmente tal forma de entretenimento seja também utilizada em dispositivos mais simples, como telefones celulares. Porém, diferente dos jogos para PCs, onde diversas empresas podem construir jogos baseados em um sistema operacional ou em uma plataforma, diferentes desenvolvedores não podem criar jogos para dispositivos móveis. O desenvolvimento de jogos para dispositivos móveis é quase sempre restrito aos fabricantes de cada modelo, sendo que os programas já vêm inclusive instalados de fábrica em cada aparelho.

Alguns dispositivos específicos, como PDAs e Handhelds, possuem sistemas operacionais proprietários que permitem a instalação de novos programas. Com isso, através de uso de compiladores específicos, é possível o desenvolvimento de jogos para estes aparelhos. Porém, neste caso, o problema é a falta de portabilidade dos jogos, dado que um aplicativo concebido para uma plataforma não pode ser utilizado em outro dispositivo que possui uma plataforma de execução diferente.

Porém, esta situação tende a melhorar consideravelmente com o surgimento de uma arquitetura padrão, e principalmente aberta, para o desenvolvimento de aplicações para dispositivos móveis, chamada J2ME – Java 2 Micro Edition [2]. Tal tecnologia consiste na simplificação da linguagem e da máquina virtual Java, de forma a mesma melhor se adequar a escassez de recursos (memória, capacidade de processamento) disponíveis nos dispositivos móveis, sendo sua criação o subsídio necessário para a concepção de jogos para tais dispositivos, tal qual acontece em computadores pessoais.

Ao contrário das aplicações que utilizam WAP, I-mode ou SMS, J2ME proporciona ao desenvolvedor o uso de uma verdadeira linguagem de programação, fazendo com que possam ser criados aplicativos que sejam instalados, por exemplo, via *download* através de um cabo serial conectado a um PC, ou via a própria rede de dados dos celulares. Esta capacidade de instalação de programas, associada à crescente utilização de dispositivos móveis, torna tal área

um mercado tão promissor para a venda de aplicações quanto a de PCs. Estima-se que em 2005 mais de 80% da Europa utilizarão telefones celulares e mais de um bilhão de pessoas usarão a Internet através de dispositivos móveis até 2003[7]. Estas previsões, associadas ao sucesso comercial que jogos eletrônicos, demonstra o grande potencial de mercado de jogos em dispositivos móveis, fazendo com que vários investidores de risco e operadoras de telefonia móvel já tem feito parcerias no intuito de investir fortemente na área[8].

Apesar do sucesso na área industrial/comercial, o estudo sobre jogos é um campo pouco explorado na área acadêmica, podendo-se dizer que havia até um certo preconceito com a área. Esta situação era criada por uma série de fatores como a falta de instituições de pesquisa interessadas no tema, a escassez de livros e congressos, bem como de especialistas no meio acadêmico. Porém, este cenário começa a mudar. De fato, pode-se dizer que poucas são as áreas da computação onde é possível se explorar tantos domínios do conhecimento quanto a de jogos. No campo da computação, podemos citar como exemplo às de computação gráfica, inteligência artificial, computação musical, redes de computadores, engenharia de *software*, dentre outros.

Dada esta multidisciplinaridade e de tantas aplicações, algumas universidades estrangeiras já apresentam pesquisas na área de jogos eletrônicos. Como exemplo, pode-se citar a University of North Texas (EUA), Institut d'Informatique de Québec (Canadá), Centre National d'Animation et de Design (Canadá), Middlesex University School of Computing Science (Reino Unido), Universisty of Teesside (Reino Unido), University of Michigan (Reino Unido), NgeeAnn Polytechnic (Cingapura), Full Sail Real World Education e Digipen Institute of Technology, onde as duas últimas já inclusive oferecem cursos completos de graduação na área.

No Brasil, o Centro de Informática da Universidade Federal de Pernambuco já a algum tempo vem investindo na área de jogos, principalmente na pessoa do Prof. Geber Ramalho, tornando-se pioneiro na América Latina a se aventurar na área. Os cursos de graduação e pós-graduação em Ciência da Computação possuem duas disciplinas na área de jogos em suas estruturas curriculares: Projeto e Implementação de Jogos e Jogos Avançados[4]. Além disto, já foram defendidas seis dissertações de mestrado referentes a pesquisas relacionadas com a área, inclusive com uma tese de doutorado em andamento. Em especial, uma das pesquisas merece especial atenção relaciona-se diretamente com este documento: a elaboração de um *framework* para o desenvolvimento de jogos para dispositivos móveis denominado wGEM[1].

Este *framework* foi concebido de modo a facilitar a criação de jogos em dispositivos móveis, através da implementação das principais funcionalidades que a maioria dos jogos possíveis com a tecnologia J2ME necessita. Porém, apesar de ser um importante avanço para a construção de tais jogos, o wGEM em sua primeira versão ainda possui uma série de funcionalidades que precisam ser melhor exploradas. Uma delas diz respeito ao suporte aos chamados Jogos Isométricos (jogos 2D ½), que até a realização deste trabalho não era possível.

Este documento trata justamente dos procedimentos realizados durante o projeto de extensão do wGEM para suporte a jogos isométricos. Tal projeto foi realizado como trabalho final da disciplina "Projeto e Implementação de Jogos", no curso de pós-graduação da Universidade Federal de Pernambuco. Nas seções seguintes a esta introdução são apresentados o wGEM, conceitos relacionados com Jogos Isométricos e questões relacionadas ao projeto de extensão do wGEM, para suporte a estes jogos. Por fim, são apresentadas conclusões sobre o projeto e apresentados alguns trabalhos interessantes como continuação deste.

2. WGEM - *Wireless Game Engine for Mobile Devices*

A evolução presenciada na área de jogos é composta por avanços em diversas áreas, dentre as quais, estão a evolução metodológica do processo de desenvolvimento e a evolução tecnológica. Esses avanços atualmente apontam para certas tendências que têm estado cada vez mais presentes no desenvolvimento de jogos em escala profissional. A indústria de software em geral

há um bom tempo já se conscientizou que o desenvolvimento de software de qualidade implica em muita disciplina e organização do processo de desenvolvimento.

Em relação à implementação de jogos, uma tendência recente é da utilização de *frameworks*, também conhecidos como motores, para a criação de jogos. Tais *frameworks* funcionam como uma estrutura básica comum a praticamente todo jogo, permitindo que novos jogos sejam definidos e implementados a partir de recursos básicos que o mesmo provê. Desta forma, basta apenas ao desenvolvedor do jogo implementar as funcionalidades específicas de seu jogo. Espera-se que um bom motor seja formado por uma série de módulos de fácil utilização e principalmente, fácil extensão.

Tal motor é um elemento essencial para os jogos devido a sua capacidade de prover uma enorme quantidade de recursos úteis aos jogos, o que reduz enormemente, portanto, a quantidade de código a ser escrito e a dificuldade de desenvolvimento dos jogos, bem como aumenta a organização final do projeto. Tudo isso acarreta, conseqüentemente, na aceleração do processo de desenvolvimento de jogos, fator crucial em um mercado cada vez mais exigente. Sendo assim, a reutilização de tais módulos proporciona não só ganho de tempo, mas também de qualidade, uma vez que a maior parte do esforço poderá ser dedicado ao jogo em si, e não mais a detalhes técnicos sobre tarefas secundárias e características peculiares à plataforma de execução do jogo, além de a qualidade daquele módulo já ter sido previamente testado, bem como sua qualidade aprovada.

Foi intuito de conseguir tais benefícios, que surgiu o projeto do *Wireless Game Engine for Mobile Devices* - wGEM, um motor para o desenvolvimento de jogos para dispositivos móveis, utilizando tecnologia *Java 2 Micro Edition* - J2ME™. Este *framework*, desenvolvido como tema de mestrado no Centro de Informática da Universidade Federal de Pernambuco, tem por objetivo maior a melhoria da qualidade e diminuição do tempo gasto no desenvolvimento de jogos para dispositivos móveis, como telefones celulares, trazendo consigo os benefícios já existentes de motores de jogos convencionais, adaptados a um ambiente escasso de recursos provido por J2ME e pelos próprios dispositivos embarcados aos quais estes jogos se destinam.

Em sua primeira versão, o wGEM permite a criação de jogos de forma mais rápida e fácil, através do reuso de sua API – *Application Programming Interface*, simplificando com isso, boa parte do desenvolvimento para aqueles que desejam desenvolver jogos para celulares. Porém, apesar de seu sucesso em relação a implementação de vários jogos, ainda existem certas extensões e aplicações interessantes para este *framework*. Uma dessas diz respeito ao suporte aos chamados jogos isométricos, suporte este ainda não provido pelo wGEM em sua primeira versão.

3. Jogos Isométricos

Para melhor compreensão das modificações realizadas na versão inicial do wGEM, faz-se necessário inicialmente entender o conceito de jogos isométricos, principalmente no que diz respeito à construção dos mapas que compõem os cenários, bem como no que tange à navegação dos personagens no jogo. Em jogos simples, o mapa é subdividido em áreas menores, chamados *tiles*, por onde os objetos animados (*sprites*) são desenhados e se movimentam. Nestes jogos, o formato destes *tiles* é retangular, fazendo com que a construção do mapa, navegação e detecção de colisão entre personagens sejam mais facilmente tratados. Tais mapas são representados por uma matriz de *tiles* - *TileMap*, onde para desenhar o mapa é apenas necessário percorrer cada linha da matriz, desenhando cada *tile* ao lado de seu antecessor, e cada linha abaixo de sua predecessora, tal qual a própria matriz.

Jogos isométricos ou 2D½[3], por sua vez, são aqueles que apresentam uma peculiaridade em relação aos *tiles* que compõem os mapas do jogo. Nestes mapas, os *tiles* são angulares (geralmente losangos), fazendo com que a disposição dos mesmos possibilite diferentes

configurações em relação ao formato geral do mapa. Os mais conhecidos formatos de mapas isométricos são: *diamond*, *staggered* e *slide*, apresentados segundo a Figura 1.

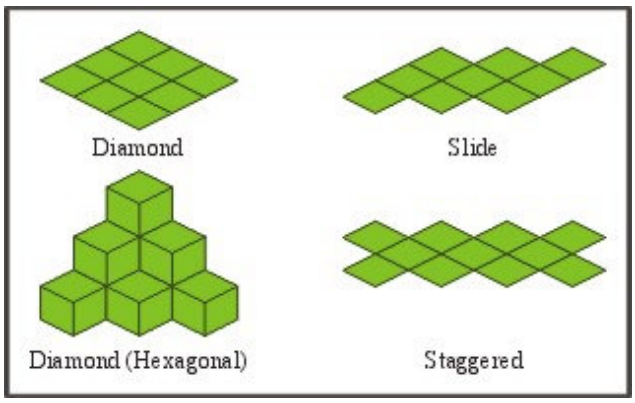

Figura 1- Tipos de Mapas Isométricos

Ressalta-se que estes mapas também são representados por um *TileMap*, de ordem 2x2, porém a visualização do mapa dependerá do formato escolhido. Tais mapas apresentam uma série de questões relevantes para quem desenvolve os jogos. As principais perguntas a serem respondidas são: dada uma posição no *Tilemap*, como encontrar a coordenada no mapa (em que posição da tela este ponto deve ser desenhado)? Dado um ponto na tela, qual a sua posição no *Tilemap*? Como devem ser os deslocamentos dos objetos de um ponto do mapa para o outro?

Para melhor ilustrar tais questões, a Figura 2 mostra dois mapas, um no formato *diamond* e outro no formato *staggered*, onde seus *TileMaps* são de ordem 3 x 3. Observa-se por esta figura, que o tipo do mapa determina onde cada *tile* deve ser desenhado. Enquanto para o mapa *diamond*, o *tile* (0,1) deve ser posto abaixo e a esquerda do *tile* (0,0), para o mapa *staggered*, o mesmo é posto abaixo e a direita. Por conseqüência, estas diferenças vão afetar diretamente a navegação dos personagens no mapa. Na Figura 2, imaginando-se que um personagem do jogo esteja sobre o *tile* (1,1), uma movimentação do mesmo para a direita representa que o personagem estará em *tiles* diferentes dentro do *tilemap*, dependendo do mapa.

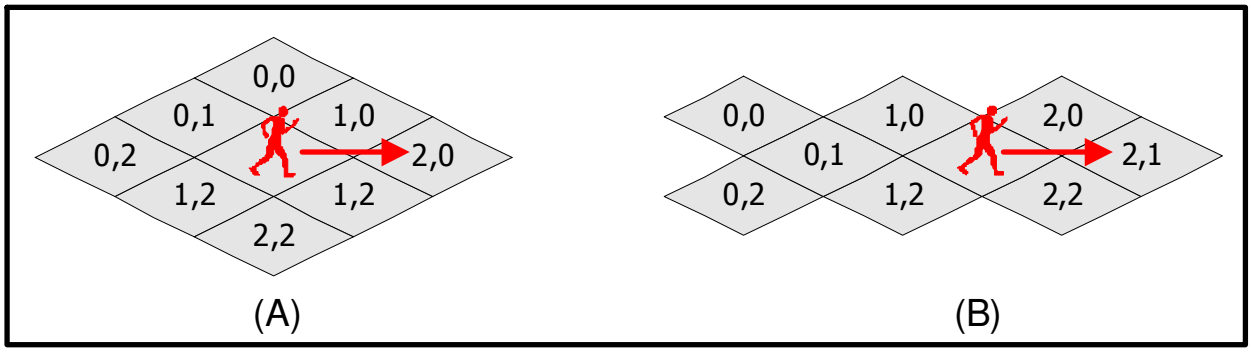

Figura 2 – Diferença de renderização e navegação entre dois mapas isométricos: (a) Diamond e (b) Staggered

4. Extensão do wGEM

A partir dos objetivos propostos, foram definidos então alguns pontos em relação ao suporte por parte do wGEM a jogos isométricos. Foram levantadas questões sobre quais mapas a serem implementados, a como mapear o teclado do celular para a movimentação dos personagens, dentre outras questões. Nas próximas sub-seções, as principais decisões do projeto são relatadas.

4.1. Limitações

Na fase inicial do projeto, algumas decisões foram tomadas de modo a facilitar o alcance dos objetivos desejados. Primeiramente, decidiu-se implementar versões para todos os mapas citados

previamente na seção 1. Em seguida, a necessidade de sobreposição dos *tiles* para a renderização do mapa implicou que houvesse alguma forma de *clipping* dos *tiles* que compõem cada mapa. Sem a utilização de transparências entre os *tiles*, haveria uma sobreposição entre os mesmos, fazendo com que o efeito desejado não fosse alcançado. Uma solução para este problema, inclusive testada, foi a realização do *clipping*, em memória, da área da figura geométrica (losango, hexágono) que realmente compõe cada *tile*. Porém, J2ME em suas bibliotecas gráficas, oferece apenas uma função para *clipping* de imagem, sendo que esta utiliza apenas retângulos para delimitar a área de *clipping*. Desta forma, para realizar um "*clip*" de uma imagem como um losango ou um hexágono, seria realizado um procedimento que decomporia a imagem em várias áreas, o que acabou tornando-se inviável pelo tempo da realização desta operação, e seu comprometimento no desempenho dos jogos. Com isso, a extensão do wGEM para jogos isométricos adequa-se apenas a dispositivos que permitam o uso de transparência para as imagens dos *tiles*.

4.2. Visão lógica da arquitetura do sistema em relação aos mapas

A partir da definição dos mapas a serem implementados, a extensão do wGEM tornou-se relativamente tranqüila, pela modularidade com que o mesmo fora projetado. A implementação dos mapas, por exemplo, estendeu classes bases já existentes no wGEM. Para melhor tal implementação, A Figura 3 ilustra parte do diagrama de classes em UML[6] dos objetos que representam os possíveis mapas no wGEM estendido.

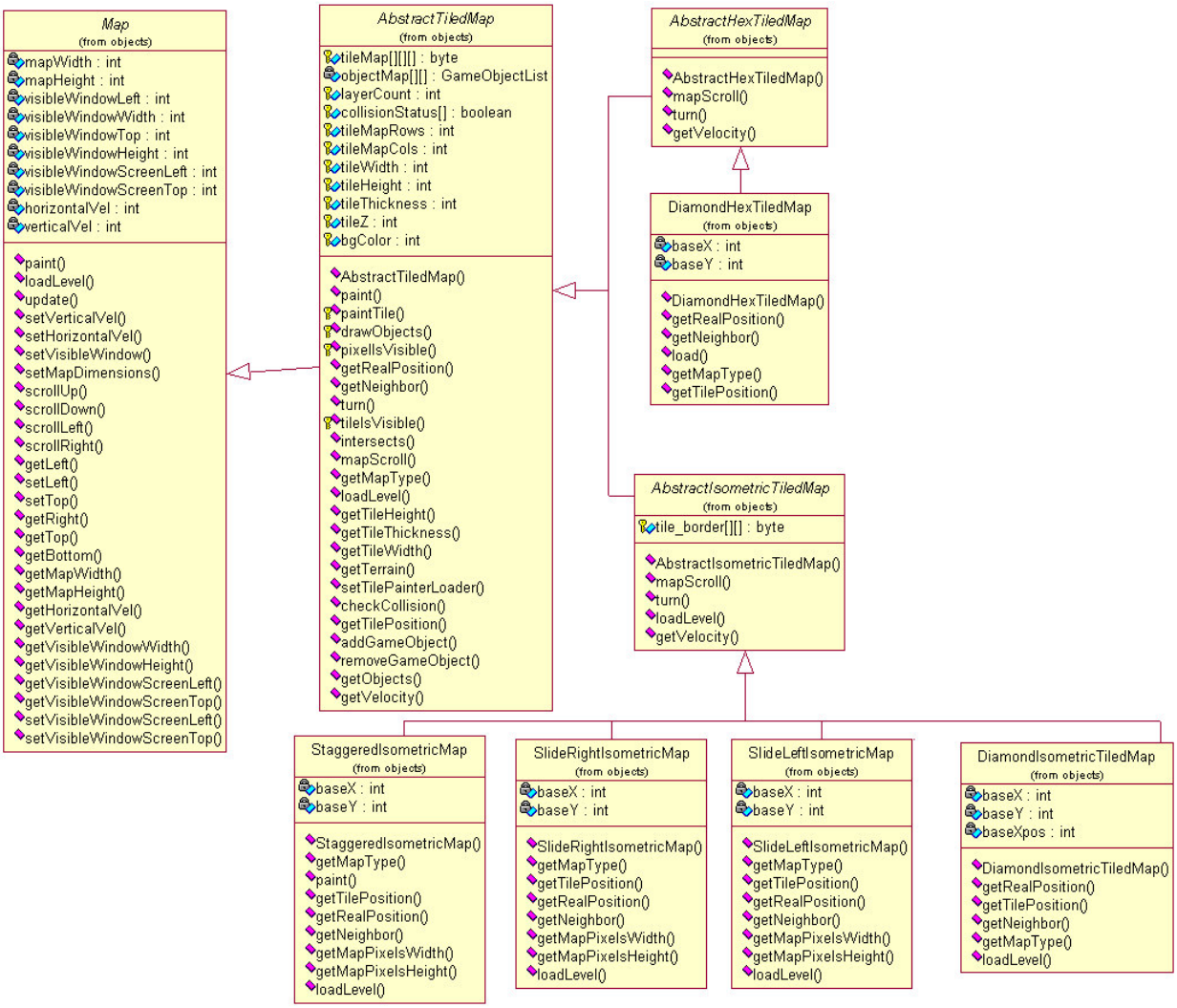

Figura 3 - Diagrama de classes em UML dos mapas

A classe Map é remanescente do wGEM original e representa as operações básicas e comuns a todo tipo de mapa, representadas por seus métodos. As demais classes foram introduzidas de modo a fornecer suporte aos jogos isométricos.

A classe AbstractTiledMap representa as operações existentes em qualquer mapa de *tiles,* seja este isométrico ou não. As funções comuns apenas aos mapas isométricos foram implementadas na subclasse chamada AbstractIsometricTiledMap, que por sua vez é a super-classe das implementações para cada mapa isométrico que se queira utilizar, que são representadas pelas classes StaggeredIsometricMap, SlideRightIsometricMap, SlideLeftIsometricMap e DiamondIsometricTiledMap. As classes AbstractHexTiledMap e DiamondHexTiledMap relacionam-se com a implementação de mapas hexagonais.

4.3. Renderização do Jogo

Em um jogo, a ordem de renderização do mapa e dos *sprites* é bastante significativa. De maneira geral, deve-se seguir uma ordem de desenho de modo a evitar problemas, como por exemplo, o corte de partes dos objetos pelo mapa. Para que isto não aconteça deve-se desenhar os *tiles* na ordem de cima para baixo na tela, e para cada linha desenhada, deve-se desenhar os objetos na ordem de suas camadas, da mais inferior a mais superior. Os objetos também devem ser renderizados a cada *tile* considerando a posição de cada objeto no mesmo.

O wGEM não oferecia tal tipo de renderização, pois em sua primeira versão, primeiro era desenhado todo o mapa, para depois então ser desenhado todos os objetos. Para resolver este problema, o motor do wGEM teve que ser alterado para não desenhar os objetos após o mapa, e sim durante a própria renderização deste. Devido a este requisito, o mapa deve ter acesso aos objetos do jogo, sendo necessário manter em cada *tile* uma lista ligada de objetos que se localizam sobre o próprio *tile*.

Foi criada a classe wGEM.objects.WalkerGameObject, que estende a classe original do wGEM para representar objetos do jogo - wGEM.objects.GameObject. A funcionalidade desta classe é manter o mapa atualizado de modo que toda vez que a posição de um objeto for alterada, como no caso de deslocamento entre *tiles*, a lista de objetos de *tiles* do mapa é atualizada para refletir esta modificação. A Figura 4 mostra o diagrama da classe, com seus principais métodos.

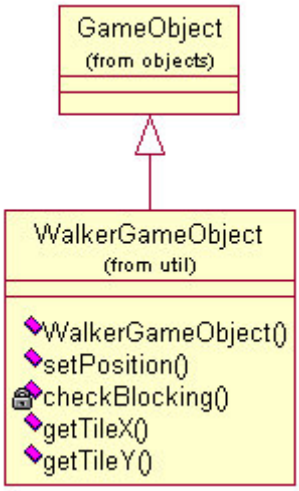

Figura 4 - Diagrama UML da classe WalkerGameObject

4.4. Detecção de Colisão

O modelo de detecção de colisão entre objetos foi outro ponto abordado e modificado no wGEM para jogos isométricos. O fator determinante para isto é que para mapas isométricos, a altura dos objetos deve ser levada em consideração para se determinar a colisão ou não entre objetos, dado que em certas situações pode haver a interseção das imagens mesmo sem que haja necessariamente uma colisão. A solução para este problema foi utilizar a projeção da altura de cada objeto para se determinar quando haverá colisão entre eles. Desta forma, só haverá colisão quando houver uma interseção das projeções de dois objetos sobre o mapa. Então foi utilizada a mesma técnica de detecção de colisão já existente no wGEM, só que aplicada à projeção do objeto no mapa. A Figura 5 exemplifica um caso onde não há colisão e as imagens dos objetos se sobrepõem e outro caso onde há colisão entre os objetos.

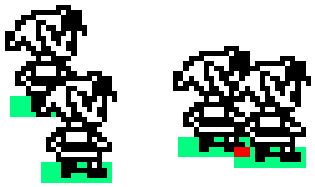

Figura 5 - Uso de projeções para determinar colisão entre objetos no WGEM Isométrico

4.5 Scrolling

O uso de mapas de *tiles* gera a necessidade de algum gerenciamento da rolagem da tela a medida que os objetos do jogo se movem. Este rolamento da tela, comumente chamado de *scrolling*, foi implementando e disponibilizado nesta versão do wGEM isométrico oferecendo recursos básicos de *scroll* da tela do jogo. Foram implementadas abordagens para realizar rolamento de um objeto, através da implementação da classe wGEM.objects.TiledMapScroll, que oferece recursos para fazer *scroll* sobre um objeto qualquer do jogo independente do mapa de jogo utilizado. A Figura 6 apresenta o diagrama UML desta classe.

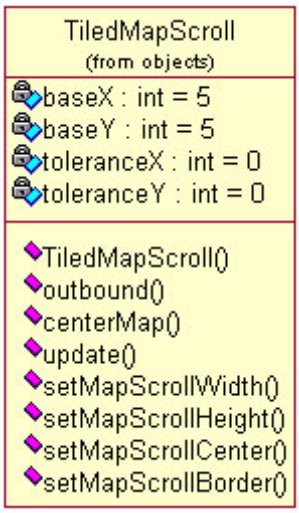

Figura 6 - Diagrama UML da classe TiledMapScroll

Esta classe permite operações como atualizar a janela visível do jogo, centrando-a em um determinado objeto através do método centerMap(), ou ainda, testar se um objeto está dentro ou fora da área visível da tela, através do método outbound(). Tratando-se de uma área delimitada, são então possíveis diferentes tipos de *scrolls* como, por exemplo: *scroll* mantendo o objeto sempre centrado na janela visível ou *scroll* que atualiza a janela visível apenas quando o objeto chegar na borda, ou ainda um meio termo entre estas abordagens.

4.6. Facing e Turning

Por fim, outra funcionalidade modificada no projeto foi as operações de *facing* e *turning* de personagens do jogo. Em um jogo isométrico, dada uma posição no mapa, um personagem pode estar posicionado em direções diferentes e se deslocar para alguma destas. *Facing* é a característica que indica qual a orientação atual de um objeto do jogo. Em jogos isométricos as direções mais usadas são oito: os quatro pontos cardeais principais que são o norte, o sul, o leste e o oeste; mais os quatro pontos intermediários, o nordeste, o sudeste, o sudoeste e o noroeste. No projeto foram então implementadas duas classes wGEM.objects.FacingObject e wGEM.util.visualContent.AnimatedSpritePainter, que armazenam informações da direção atual de um objeto e de movimento de giro, permitindo com isso que a partir da entrada do jogador, o *sprite* possa ser modificado para refletir a nova direção para onde o personagem esteja direcionando-se. A Figura 7 mostra o digrama UML destas classes.

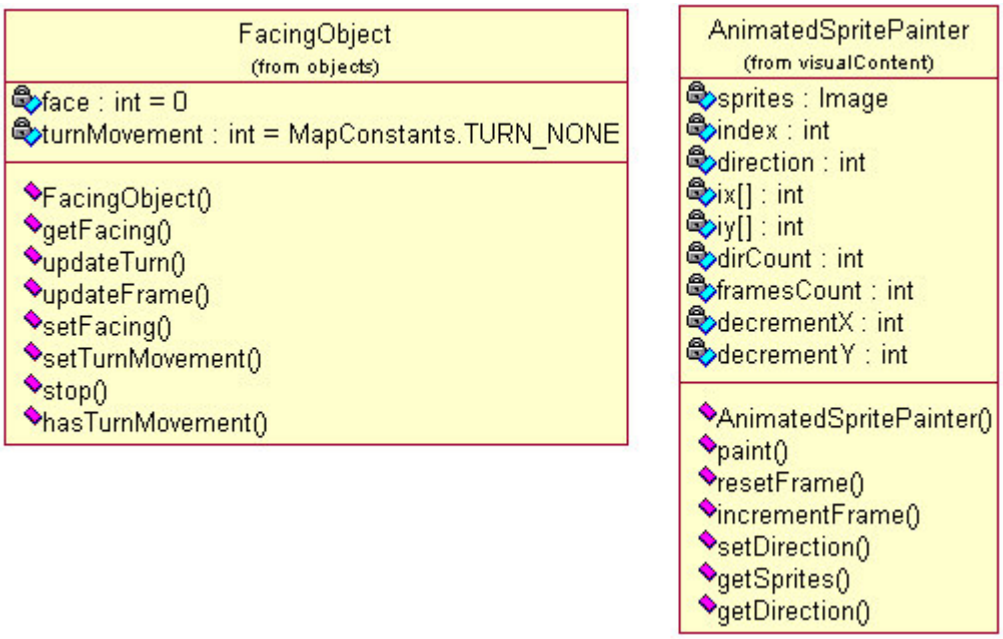

Figura 7 - Diagrama UML da classe FacingObject e AnimatedSpritePainter

As informações de direção atual e movimento de giro variam de acordo do tipo de mapa e estão todas definidas como constantes inteiras na classe wGEM.objects.MapConstants. Tomou-se ainda como decisão de projeto fazer com que o objeto sempre deva primeiro se posicionar para a direção, para só então se deslocar.

4.7. Altura e Thickness

Altura e *thickness* são atributos que simulam a altura de objetos em um mapa isométrico. *Thickness* é o parâmetro que dá a noção de espessura da base um *tile*, enquanto sua altura ou do objeto dá a noção de extensão sobre a superfície do mapa.

Para a extensão do wGEM isométrico foram criadas classes que possibilitam a implementação de objetos com espessura e altura. O uso de altura criou a possibilidade de objetos ficarem a frente dos personagens do jogo, e modificarem a apresentação final, quando, por exemplo, o personagem passar por tráz de uma parede ou muro.

5. Estudo de caso – *The Treasure Hunt*

Após realizar todas as implementações necessárias a dar suporte a jogos isométricos no wGEM, foi então implementado um jogo, utilizando um mapa isométrico *diamond,* de modo a validar o trabalho realizado. No cenário do jogo proposto, chamando então de “The treasure hunt”, o

jogador assume o papel de um duende que precisa achar chaves de baús, abrí-los e acumular riquezas. As chaves estão espalhadas pelo mundo e vigiadas por outros duendes que vão tentar impedir o duende de adquirir as chaves. O jogador para se defender, pode lançar também, bolas de energia, que podem congelar os duendes por um tempo limitado. O jogador tem um número limitado destas bolas. O jogo acaba quando todos os baús do mapa são abertos pelo jogador, fazendo com que este então passe de fase. A Figura 8 apresenta algumas imagens do jogo, apresentados no simulador de um telefone celular colorido.

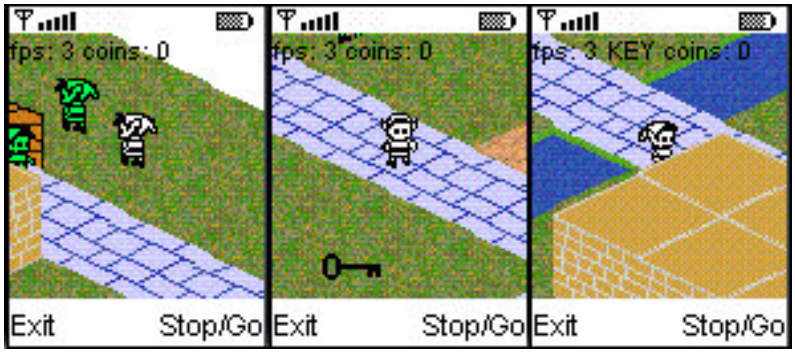

Figura 8 – O jogo "*The Treasure Hunt*"

A implementação do jogo foi feita com a ferramenta Jbuilder 6, utilizando o módulo MobileSet 2.0 para emulação do celular. Nos testes realizados, a maior preocupação se deu em relação ao desempenho do jogo, devido a maior complexidade para renderização do mapa e movimentação do personagem. Os resultados mostraram-se bastante dependente do perfil da máquina onde o emulador fosse executado. Os resultados em termos de variaram de 3 frames por segundo em um PC com processador K62 com 128Mb de memória RAM, até 14 frames por segundo, em um computador com processador Pentium 3 e 512Mb de memória RAM.

Devido a esta não conformidade de ambiente, foi realizado como teste final do jogo em um telefone celular do grupo de pesquisas da Motorola, no Centro de Estudos Avançados do Recife – C.E.S.A.R.[5], para avaliação do desempenho do jogo num dispositivo real. Nestes testes, os resultados mostraram bastante satisfatórios no que tange a velocidade de renderização do jogo, com taxa de renderização de 12 frames por segundo, o que não comprometeu em nada a jogabilidade do mesmo.

6. Conclusões e Trabalhos futuros

Após a realização deste projeto, vimos que a construção de jogos para dispositivos móveis tende a sofrer um impulso muito acentuado com a chegada da plataforma J2ME, e que a existência de um motor como o wGEM é um excelente facilitador para o ganho de tempo e qualidade na implementação de boa parte destes jogos.

A extensão conseguida através do projeto relatado neste trabalho serve para aumentar ainda mais, a gama de possibilidades de jogos a serem suportados por este motor.

Através deste trabalho, o wGEM permite que sejam implementados jogos utilizando as mais diversas configurações de mapas isométricos, bem com opções de *scrolling* de objetos do jogo.

Apesar de muito ter sido conseguido na implementação deste projeto, há trabalhos que ainda podem continuar a melhoria do mesmo. Dentre alguns, podemos citar:

a) Criação de um editor de mapas para o jogo implementado, pois atualmente, a edição de mapas dos níveis do jogo é realizada via editores de texto, fazendo com que o criador necessite entender todo o formato do mapa, fato este que não é trivial;

b) Otimização dos algoritmos de renderização dos mapas utilizados no projeto;

c) Análise de desempenho em relação a outras possíveis plataformas de desenvolvimento para dispositivos móveis.

7. Referências

[1] PESSOA, C. A. C. **wGEM: Um framework de desenvolvimento de jogos para dispositivos móveis**. Dissertação de Mestrado. Centro de Informática – UFPE, 2001.

[2] J2ME(TM) - Java 2 Platform, Micro Edition, http://java.sun.com/j2me/ (10/04/2002)

[3] **Isometric and Tile-based Games. http://www.gamedev.net/reference/list.asp?categoryid=44 (03/04/2002)**

[4] **Projeto e Implementação de Jogos**, Cin/Ufpe,http://www.cin.ufpe.br/~game/ (03/03/2002)

[5] **Centro de Estudos Avançados do Recife – CESAR**, http://www.cesar.org.br (01/08/2002)

[6] JACOBSON, J et al. **The Unified Modeling Language User Guide**. Addison-Wesley. 475p. 1998

[7] BRIGGS, J. **J2ME: The next major games plataform?.** Disponível na Internet. URL: http://www.javaworld.com/javaworld/jw-03-2001/jw-0309-games_p.html. Consultado em 03 de agosto de 2001.

[8] **WapDrive,** ***Buzztime announces the success of its WAP game.*** Disponível na Internet. URL: http://www.wapdrive.com/DOCS/wap_news/02_17_2001.html. Consultado em 4 de agosto de 2001.